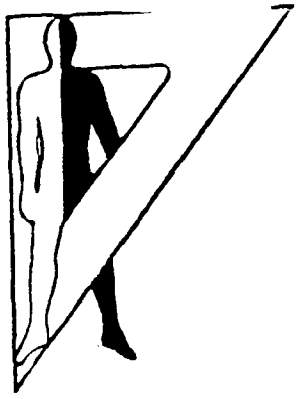

Technical Note 3-92

SYSTEM FOR GENERATING DYNAMIC VIDEO IMAGERY FOR HUMAN FACTORS RESEARCH

Dale R. Shires
Franklin F. Holly
C. Alec Chang
Joseph Mazurczak
Salvatore P. Schipani
Phillip G. Harnden

February 1992
AMCMS Code 61110274A0011

Approved for public release
distribution is unlimited.

U.S. ARMY HUMAN ENGINEERING LABORATORY
Aberdeen Proving Ground, Maryland

®Apple is a registered trademark of Apple Computer, Incorporated.
™Macintosh is a trademark of Apple Computer, Incorporated.
™THINK C is a trademark of Symantec Corporation.
™NuVision is a trademark of Perceptics Corporation.

Destroy this report when no longer needed.
Do not return it to the originator.

The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial products.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Note 3-92	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Human Engineering Laboratory 6b. OFFICE SYMBOL (*If applicable*) SLCHE	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Aberdeen Proving Ground, MD 21005-5001	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 6.11.02 / PROJECT NO. 1L161102B74A / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)

System for Generating Dynamic Video Imagery for Human Factors Research

12. PERSONAL AUTHOR(S)

Shires, D. R.; Holly, F. F.; Chang, C. A.; Mazurczak, J.; Schipani, S. P.; Harnden, P. G.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Final	FROM _____ TO _____	1992, February	43

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	
12	7		image processing, computer graphics, dynamic imagery, video systems, visual imagery, human factors, real time
14	2		

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

A computer-controlled system was developed which provides the capability to generate processed dynamic imagery for human factors research. Processing of the video may involve any operation such as enhancing, degrading, or adding overlays. This system may also be used during experimental trials to perform data collection and score subject responses.

The system is comprised of several components: a computer, an image processor, two time lapse video cassette recorders, and a custom built VCR control circuit. When possible, processing is performed as the video is being displayed to a subject. When processing complexity is too great, the video is processed before experimentation. The input to the system is imagery on a video tape. The imagery is processed frame by frame and the results are stored on the output video tape. Processing appears to be performed in real time when this resultant tape is played at the regular speed.

The flexibility and cost effectiveness of the system make it an ideal developmental tool, test-bed, and simulator for the processing of dynamic visual imagery. The hardware and software for implementing the system are discussed in detail.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified

22. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*)	22c. OFFICE SYMBOL
Technical Reports Office	(301) 278-4478	SLCHE-SS-TSB

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

AMCMS Code 61110274A0011 Technical Note 3-92

SYSTEM FOR GENERATING DYNAMIC VIDEO IMAGERY FOR HUMAN FACTORS RESEARCH

Dale R. Shires
Franklin F. Holly
C. Alec Chang
Joseph Mazurczak
Salvatore P. Schipani
Phillip G. Harnden

February 1992

APPROVED:

T. K. H. WONG
LTC, TC
Acting Director
Human Engineering Laboratory

U.S. ARMY HUMAN ENGINEERING LABORATORY
Aberdeen Proving Ground, Maryland 21005-5001

CONTENTS

SYSTEM FOR GENERATING DYNAMIC VIDEO IMAGERY FOR HUMAN FACTORS RESEARCH

INTRODUCTION

The U.S. Army Human Engineering Laboratory has developed a computer-controlled system designed for examining human performance in tasks that require operators to extract information from moving images on visual displays. This system was developed because of the frequent need to use dynamic imagery rather than static imagery when performing human factors research. This system has two major functions. The first paramount function is to generate processed dynamic imagery that can be used in a wide range of experiments designed to test human perception of nonstatic imagery. The second function of the system is to act as the data-collection device during experimentation.

Unfortunately, processing video imagery at real time rates (30 frames per second) is computationally intense. When the worst case performance profile of an algorithm running on a certain computer is greater than 1/30 second per video frame, real time or live processing is impossible. At times, improved algorithms with lower time complexities or computers with tailored architectures will allow for real time video processing. The problem with customized systems is that they are very expensive and are not universal enough to accommodate new algorithm development. Newly developed procedures and functions will continually require extensive hardware modifications.

The computer-controlled system avoids inherent problems associated with customized hardware by using a process known as simulated real time processing. In this procedure, a general purpose image processor and a computer perform the desired operation on the dynamic imagery one frame at a time. Only those video-processing operations not feasible in real time during the actual experimentation are performed in this frame-by-frame manner. The resultant frames are recorded frame by frame on an output videotape. When the output tape is played at the regular 30-frames-per-second speed, the video appears as if it is being processed in real time. During this frame-by-frame process, computer-generated graphics and researcher-controlled stimuli, such as hazard objects or targets, may be added to the video.

SYSTEM UTILITY AND LIMITATIONS

Use of the computer-controlled system is appropriate whenever one wishes to study the perceptual ability of subjects while viewing imagery having a fixed path and speed. In a typical application of this system, real world dynamic video imagery is generated by mounting a video camera on a moving vehicle and then recording the imagery onto videotape. Synthetically created imagery of objects, such as targets or hazards, may then be overlaid onto the videotape and, if required, other image-processing operations can be performed to enhance or degrade video resolution. Use of this system is not appropriate when the subject is required to affect the imagery (e.g., to change its path, speed, etc.) in real time.

This system provides a cost-effective way to generate processed dynamic imagery. The total cost for all parts of the system is approximately $40,000. The system is also extremely flexible. New processing features may be added in software, thus eliminating the need for hardware modifications.

Simulation in perception experiments using this system has some advantages compared with field research. First, confounding factors can be excluded so that experiments can focus on the parameters to be studied. Second, true values can be known for validation and experimental result scoring purposes (as opposed to the errors inherent in attempting to obtain "ground truth" data in field experiments).

In addition to its utility for the human factors research community, this system may be used by design engineers to examine what processed imagery would look like before going to the expense of developing special purpose computer hardware to run proposed algorithms in true real time. After using this system to determine the correct algorithm and optimal processing schema, engineers may develop appropriate architectures for real time processing.

When the complexity of an algorithm is sufficiently great, the controlling computer may not be able to complete video processing in an acceptable period of time. In this case, this system can be modified to digitize and store all frames of the input video in advance. These files may then be processed by another computer with greater computational power. The system may then be configured to read the resultant processed files and to place them on the output videotape.

This system is designed to work only with gray scale images. Color processing would require composite-red, green, blue (RGB) video encoders/decoders and extensive revisions of existing software.

SYSTEM OVERVIEW

The system is controlled by an Apple Macintosh IIfx computer running under C and includes the following additional components:

1. A NuVision image processor from Perceptics Corporation.

2. Two Victor Company of Japan (JVC) time lapse video cassette recorders (VCRs) model BR-9000U. One VCR is for video playback and the other VCR is for video recording.

3. A digital time base corrector, model FA-300, from FOR-A Corporation.

4. A custom-made circuit for Macintosh control over VCR frame advance.

5. A video monitor.

HARDWARE

Figure 1 illustrates the system components and hardware connections. "Video out" from the playing VCR is connected to "video in" on the time base corrector. Video out from the time base corrector is connected to the video in terminal No. 1 on the NuVision system. The green video out port on the NuVision system is connected to video in on the recording VCR. Video out from the recording VCR is connected to the video in terminal on the monitor. Pin 1 from the mini 8-pin connector on the Macintosh printer serial port is

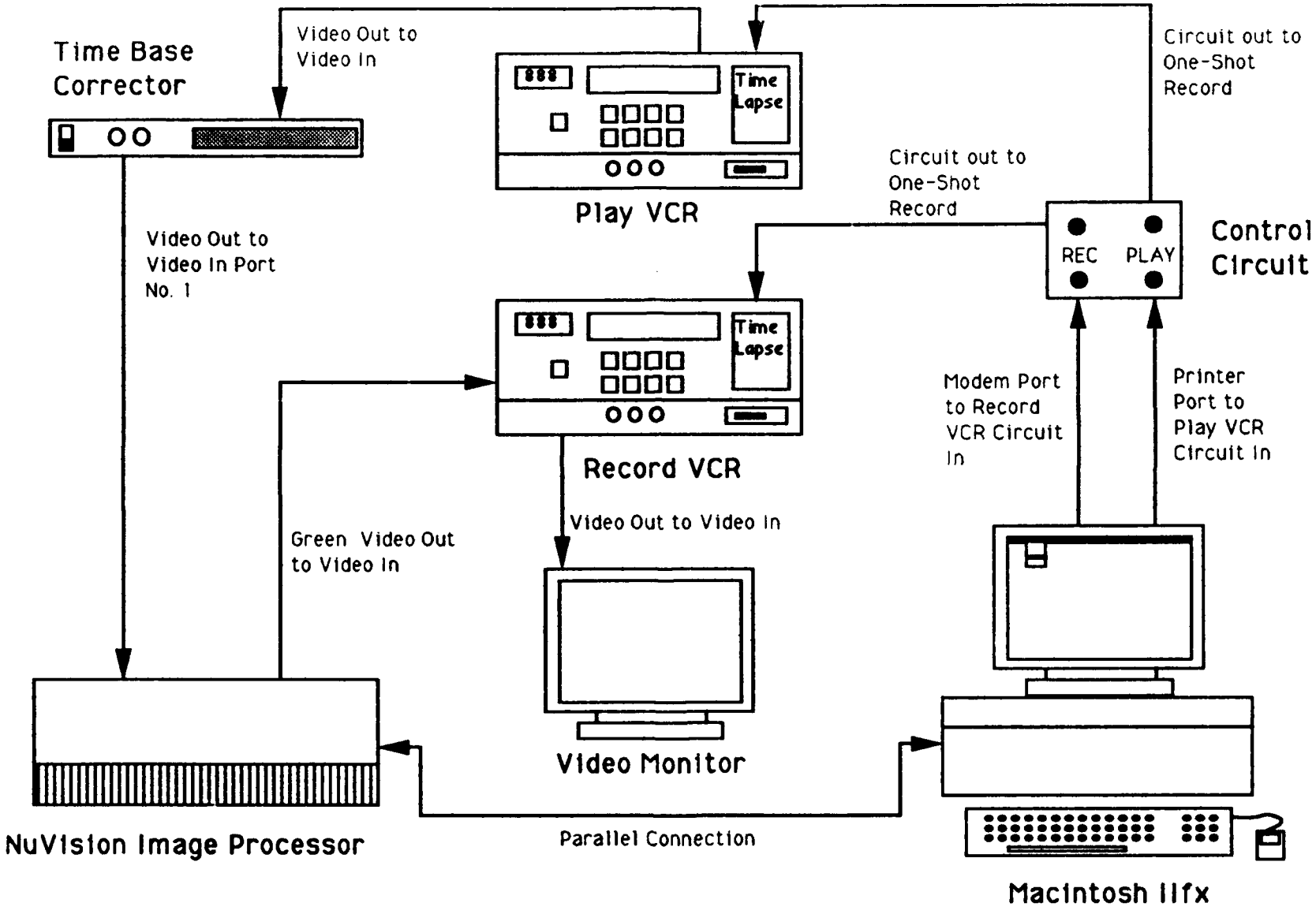

Figure 1. System for generating processed dynamic imagery.

connected to the input on one circuit. The output from this circuit is connected to the one-shot record terminal on the back of the playing VCR. Pin 1 from the mini 8-pin connector on the Macintosh modem serial port is connected to the input of the other circuit. The output from this circuit is connected to the one-shot record terminal on the back of the recording VCR.

Macintosh IIfx

The computer controlling this system is a Macintosh IIfx manufactured by Apple computer. The image processor used in this system is designed to work in conjunction with any Macintosh II family computer to form an imaging work station. The computer has 8 megabytes (MB) of memory and a 40-MHz clock speed and was chosen because it is the most advanced computer in the Macintosh II line. Memory capacity and speed should be important considerations when selecting the controlling computer since it must perform all operations not available on the image processor. Even marginal increases in computer efficiency will dramatically reduce the total time required to complete frame-by-frame video processing.

Perceptics NuVision Image Processor

The image processor contains general purpose digital signal processors (DSPs) and can perform some functions, such as gray scaling, in real time. The optional pixel processor board is required to perform these real time operations. The image processor is controlled from the Macintosh computer by making calls to the C software library provided with the image processor. All analog-to-digital and digital-to-analog video conversions are performed using the image processor.

The output video signal of the image processor must be configured to RS-170 format for compatibility with the VCRs. This is accomplished by changing the jumper on the video interface board from RGB to RS-170. Details about performing this jumper change can be found in the NuVision manual.

JVC Time Lapse VCRs

The JVC time lapse VCRs offer many different record and play speeds. For this system, the one-shot record and play are used. An internal switch (SW301) must be set for one-shot record and play to function. Refer to the VCR service manual for more information about this procedure. The record and play modes on both VCRs must be set to the "manual" position. The playing VCR is set to play, and the recording VCR is set to record. The VCRs will remain inactive until receiving control signals from the custom circuit interface.

Time Base Corrector

A time base corrector is used to stabilize the video signal coming from the playback VCR and to provide a clean synchronization signal to the image processor. This allows the image processor to complete frame digitization.

Custom-Built Control Circuit

A diagram of the custom-built control circuit is shown in Figure 2. Two custom-built control circuits are required, one for the recording VCR and one for the playing VCR. This circuit provides a momentary closure to ground which activates the one-shot record and playback on the VCRs. The closure duration should not exceed 1/30 second to avoid recording or playing more than one frame of video.

Input to this circuit comes from the Macintosh computer serial ports. The data terminal ready (DTR) signal output (pin 1 on the mini 8-pin connector) provides the control command to the VCR. Asserting the DTR on instruction (see the VCRControl.c file in the appendix) initiates a short duration pulse (approximately 30 ms) to advance the indicated VCR by one frame.

The HCPL-2211 optocoupler logic gate provides circuit isolation because of electrical incompatibility between the Macintosh computer and the VCR. The N74123 sets the momentary closure duration. The N7407 is used as the closure driver buffer for the VCR and a visual light emitting diode (LED) closure duration indicator.

Video Monitor

A composite video monitor is used to observe the actual frame-by-frame image processing. The standard RGB monitor supplied with the NuVision system may be used during experimental trials.

SOFTWARE

All software for this system is written in C. The THINK C version 3.01 development environment from Symantec, Incorporated is used. All code written using this programming environment may take full advantage of the Macintosh graphical interface. Code for the NuVision program is included with the image processor and is copyrighted by Perceptics, Incorporated.

The modules for simulated dynamic video production and experimental data collection and scoring are designed as "hooks" into the main NuVision software. They are invoked by way of a "Special" menu in the NuVision interface. This menu is designed to allow user written functions to be easily called from the NuVision interface. Programs called from the "Special" menu are designed to be more or less independent from other NuVision routines, but may call upon other image processing modules and libraries for functions they may require. For example, if a function added to the "Special" menu requires the ability to do a frame digitization, it may use the DoGrab(...) function located in the Digitize.c file supplied with the NuVision system.

Full details about software integration methodology using the NuVision system are included in the NuVision system manual. The methods used to set certain parameters in this system, such as the number of frames to process, are not described. Most often, these inputs to functions will be specified through the use of standard Macintosh interfaces such as controls and dialog boxes. A sample dialog box used to set parameters for one implementation of this system is shown in Figure 3.

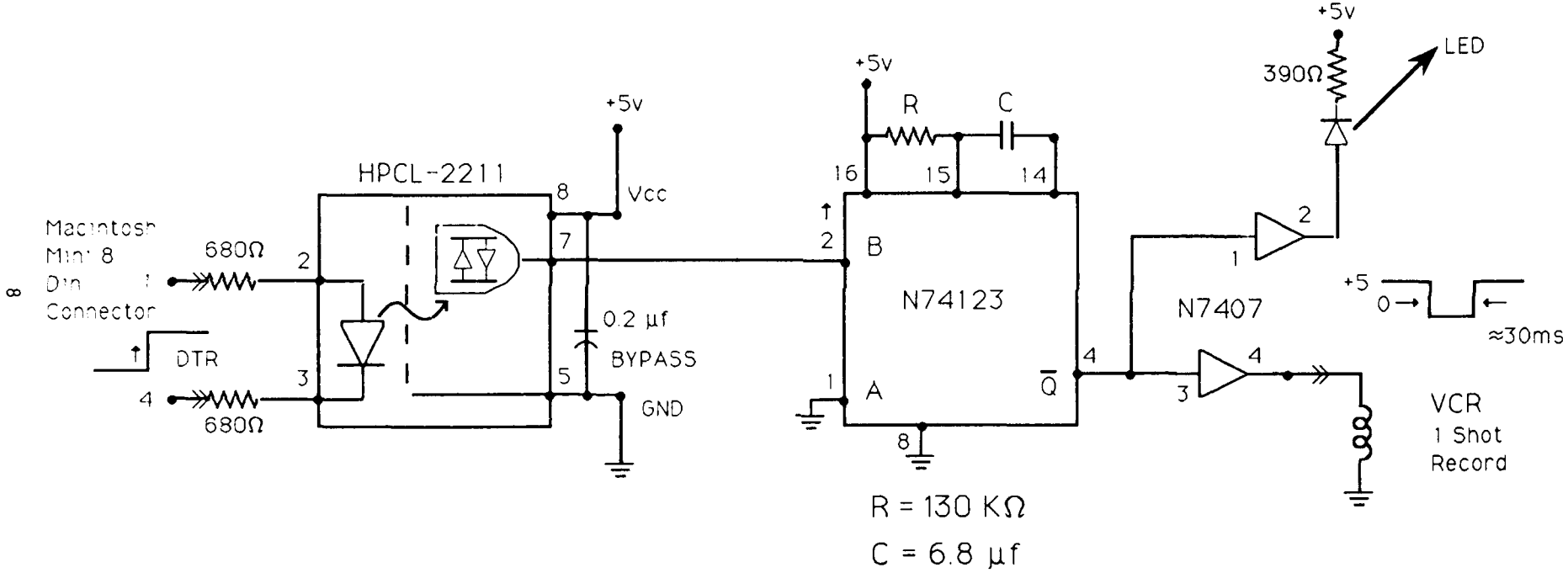

Figure 2. Custom-built VCR control circuit

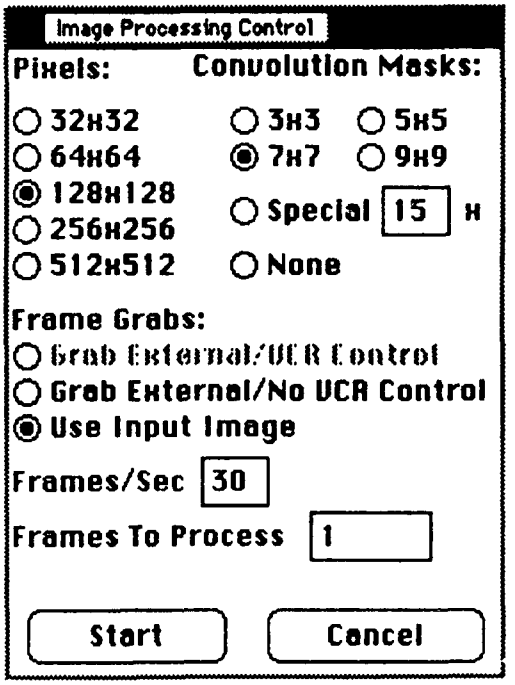

Figure 3. Example parameter setting dialog box.

Example code for this system is included in the Appendix. There are seven modules in the Appendix. The modules and their contents are in lexicographical order:

1. DataCollection - This contains the main controlling function for data collection during experimentation.

2. ImageDegradation - This contains a routine to degrade video. The example function used reduces video quality by reducing the grid sample size on a 512 x 512 pixel digital image.

3. MacintoshHeaders - This contains the most commonly used declarations and definitions for Macintosh programming.

4. MiscUtilities - This contains miscellaneous utilities for video sampling, Macintosh computer to NuVision data transfer, and so forth.

5. SystemDefines - This contains basic definitions for image size and gray scale bit depth.

6. VCRControl - This contains the code that controls the VCRs. All VCR control is effected by making state changes on the serial ports of the Macintosh computer.

7. VideoGeneration - This contains the main controlling function for processing video frame by frame.

Each function is preceded by a documentation block that details function purpose, input and output, and variable usage.

Video Processing and Tape Generation Control

The function GenerateVideo() in the VideoGeneration module is a typical main controlling function for processed video generation. There are two parameters passed to the function. The parameter numberOfFrames specifies how many video frames are to be processed from the input tape. The sampleSize parameter indicates the grid sampling size to use for degrading the video frames.

The process control of the function is straightforward:

1. A call is made to SwitchToRS170() to switch the video output signal from RGB to RS-170 format.

2. VCRs are prepared for use by a call to InitializeVCRs(). This function opens the serial ports connected to the VCR control circuit.

3. Memory is allocated for image storage with the call to AllocateMemory(...).

4. The main processing loop is entered. This loop is executed for each frame to be processed.

5. The GrabAnImage() function digitizes the video frame on the playing VCR.

6. The image intensity values in the video buffer on the NuVision are copied into the Macintosh computer memory with the call to ReadImageIntensityValues(...).

7. The playing VCR is advanced by one frame with the call to AdvanceVCR(...). This procedure toggles the DTR on and off which directs the control circuit to advance the VCR one frame.

8. The image read from the NuVision system is degraded with a call to BlockyImage(...).

9. The processed image is written back into the NuVision video buffer with a call to WriteImageIntensityValues(...).

10. The recording VCR records the processed image and advances one frame with the call to AdvanceVCR(...).

11. Control returns to Step 4 if there are more frames to process.

12. All memory used for image storage is released with the call to FreeMemory(...).

13. VCR control is terminated with the call to CloseVCRs(). This closes the serial ports connected to the VCR control circuit.

System Use in Experimentation and Data Collection

A typical experiment using this system will require a subject to indicate where and when one perceives some object on processed dynamic video. Obtaining this information is relatively easy because of the graphical interface of the Macintosh computer. A cursor is placed on the screen which the subject can move with a mouse. The subject then places the cursor at a desired position and presses the button on the mouse. This action generates an interrupt known as a "mouse down" event. The computer passes two pieces of information to the controlling software during one of these events: (1) the time the mouse button was pressed, and (2) the position of the cursor on the screen at the time the button was pressed. The cursor location is specified in horizontal and vertical screen coordinates (x,y) relative to the transparent overlay window on the monitor connected to the NuVision system. The time the button was pressed is given in the number of ticks since the computer was powered on. The tick counter is the heartbeat of the Macintosh computer and is controlled by the vertical retrace hardware. When the Macintosh computer is first started, the tick counter is set to 0. The monitor on the Macintosh computer is refreshed 60 times a second (60 Hz). Each time the video beam is moved from the bottom to the top of the screen, an interrupt is generated. The tick counter is incremented by 1 at each interrupt.

In scoring subject response data, the time returned by a mouse down event must be corrected since it relates only to the elapsed time since the system was powered on. Both subject response data elements and the truth tables with which they are compared must be based on a time relative to the start of each experimental test. The following method was devised to ensure a common base time for all experimental trials. This base time can also be used to determine a video frame count since the NuVision system does not provide this feature.

During video production, a series of black frames (i.e., in 8-bit gray scale video all pixels are set to 0) is recorded on the output tape. A 5-second lead of black video is usually desirable. At the end of these frames, a white frame (all pixels set to 255) is recorded. The video imagery on the input tape is then processed, and the frames are recorded after the white frame on the output tape.

During experimentation, video is played at normal speed through the image processor. The image processor is configured to pass all incoming video through the pixel processor. The accumulator section of the pixel processor continually monitors all incoming video frames. The total value of all pixel intensities in a frame is stored in registers dedicated to the accumulator. A small function polls these registers comparing their contents to a preset threshold value that indicates a frame of "white" pixels. The function stops polling the registers when this value is reached. The data collection software then records the tick count at the time the white frame occurred. This tick count becomes the base time or time = 0 for the experimental trial. All subsequent times of subject responses are in tick counts but are relative to the time the white frame occurred.

For example, if the white frame occurred at tick count = 179361 and the subject clicked the mouse button at time = 183000, then the subject clicked the mouse 3639 ticks or 60.65 seconds after the white frame base time. Use of this white frame gives a common time by which all experimental trials may be judged.

Frame counts are also easily obtained if base time is known. Simply multiply the time from the white frame by 0.5 since there are 30 frames for every 60 ticks. In the previous example, 3639 ticks = frame 1819 from the base white frame.

The function CollectData() demonstrates how subject responses are recorded during experimentation.

1. Memory for storing subject responses is allocated by the call to AllocateMemory(...).

2. The processed video being played on the input VCR is displayed live to the subject with the call to StartLiveVideo().

3. The WaitForWhiteFrame() function is then called. This function will wait until the white frame on the video is reached.

4. The base time is recorded as soon as the white frame passes.

5. The main loop is entered. Each time the subject clicks the mouse button, the location of the cursor and the time relative to the base time are recorded. The loop exits and data collection ends when the experimenter presses any key.

6. Live video display is cancelled with the call to StopLiveVideo().

7. Subject responses may now be scored or saved to file for later scoring.

8. All memory used is released with the call to FreeMemory(...).

Truth tables for judging experimental performance are also constructed using this base time concept. Time in these tables may be expressed in frame counts or tick counts. Truth table construction for computer-generated objects is straightforward since screen positions and frame counts are already known. Truth table generation for other objects having fixed paths and speeds in the video is more complicated but can be done very accurately using the following method.

The experimenter makes two passes through the unprocessed videotape. In pass one, the experimenter views the tape and clicks the mouse when the object in question is distant from the camera. The software freezes the picture on the screen allowing one to accurately position the cursor exactly in the center of the object. A subsequent mouse click records the position of the object on the screen and reactivates the live video. The procedure is repeated throughout the length of the video for each object of interest in the study. The same procedure is used in pass two except this time, the objects are allowed to come closer to the camera. The equation describing the slope of the path traversed by the object can now be written since two locations are known. These equations, in conjunction with equations relating world distance to screen location, can be used to accurately calculate the coordinates of the object in all intervening frames in which it appears.

The function CollectData() will allow experimenters to gather data for truth table generation. Only a few minor changes are required. On the first mouse down event, the function StopLiveVideo() is now called and the time is recorded. The experimenter then moves the cursor to the appropriate position. The second mouse down event records the position of the cursor and reactivates live video with a call to StartLiveVideo(). All subsequent pairs of mouse down events would be handled in this manner.

EXPERIMENTAL PARADIGMS AND APPLICATIONS

This system has proved useful in two research areas. First, target recognition and identification have been studied using this system. Video imagery of a highway with rounded cones or mounds randomly placed on it was processed to degrade resolution. During processing, computer-generated confusion objects were added to the video. These objects were similar to the mounds, but were created to appear and behave like flat objects on the road. Subjects were then presented the video that was further degraded in real time by reducing gray levels. Their task was to position the cursor on the hazards that they recognized as three dimensional and then click the button on the mouse. Their performance was then scored by comparing their responses with predefined truth tables.

Second, reduced data transmission techniques using simulated optic flow are being studied. Simulated optic flow attempts to simulate a series of video frames based on one frame. One frame of motion video for every 90 frames is digitized from the playing VCR. Based on the imaging geometry of the camera system used during recording, equations were written to approximate how the next subsequent 90 frames of video would appear. This system uses these equations to generate and record each of these simulated frames. The resultant video indicates the correctness of the model and suggests which parameters may need adjustment. Minute changes to fine tune these parameters are effected through software changes only, and a new output video is generated for each change. Model optimization is ongoing.

APPENDIX

SOURCE CODE

```
/*      Module: DataCollection.h

        This module contains the prototypes for functions available in
        the DataCollection.c library.

        Prototypes are listed in lexicographical order.

*/


/* Record subject actions during experimental trial */
void CollectData(void);
```

```
/*    Module: DataCollection.c

      This module contains the main routine that controls data collection
      during experimentation.  A structure is also defined in which
      information about the subject's responses can be stored.

      Functions are listed in lexicographical order.

*/


#define MAX_SUBJECT_RESPONSES       2000  /* Allow up to 2000 responses */


#include "MacintoshHeaders.h"
#include "SystemDefines.h"
#include "DataCollection.h"


/* This is the structure that will store subject responses during
   the experiment. */

typedef struct
      {
      Point mouseClickWhere;  /* record mouse location */
      long mouseClickTime;    /* record mouse click time */
      } SubjectResponse;


/* Local function prototypes (in lexicographical order): */

static void AllocateMemory(SubjectResponse *subjectClicks);

static void FreeMemory(SubjectResponse *subjectClicks);


/* external global variables */

extern WindowPtr  overlayWindow;



/************************************************************
Function AllocateMemory
************************************************************
1. Purpose: Allocate memory in which to store subject responses.
2. Inputs:  None.
3. Returned values:
            subjectClicks - An array of records for storing subject
                  responses.
4. Local variables:
            None.
5. Global variables (and how modified):
            None.
************************************************************/

static void AllocateMemory(subjectClicks)
      SubjectResponse *subjectClicks;
```

```
        {

        subjectClicks = (SubjectResponse *)malloc(MAX_SUBJECT_RESPONSES *
                sizeof(SubjectResponse));

        }       /* end AllocateMemory */


/*************************************************************
Function CollectData
*************************************************************
1. Purpose: Display a tape to the subject and record his or her actions.
            The video is passed through the image processor and the
            base time is recorded when a white frame is recognized.
            A subject response is initiated by pressing the button on the
            mouse.  The experimenter ends the study by pressing any key.
2. Preconditions:
            An output tile must be selected.
3. Inputs:  None.
4. Returned values:
            None.
5. Local variables:
            done - TRUE if the experimenter presses a key, FALSE otherwise.
            baseTime - Time at which the white frame occurs.
            theEvent - Data on what event has occurred.
            numberResponses - Counter to keep track of number of subject
                  responses.
            subjectClicks - An array of records to keep track of all
                  subject responses.
6. Global variables (and how modified):
            None.
*************************************************************/

void CollectData()
        {
        SubjectResponse *subjectClicks;
        Boolean done = FALSE;
        long baseTime;
        EventRecord theEvent;
        int numberResponses = 0;
        GrafPtr savePort;


        /* set the current port to the subject's monitor */

        GetPort(&savePort);
        SetPort(overlayWindow); /* allows us to get local mouse coordinates */

        SwitchToRS170();

        AllocateMemory(subjectClicks);

        StartLiveVideo();

        WaitForWhiteFrame();
        baseTime = TickCount();
        while (!done)
```

```
        {
        if (GetNextEvent(everyEvent, &theEvent))
              {
              switch(theEvent.what)
                    {
                    case mouseDown:
                          GetMouse(&mouseLocation);
                          subjectClicks[numberResponses].mouseClickWhere =
                                mouseLocation;
                          subjectClicks[numberResponses].mouseClickTime =
                                TickCount() - baseTime;
                          ++numberResponses;
                          break;
                    case autoKey:
                    case keyDown:
                          done = TRUE;
                          break;
                    default:
                          break;

                    }
              }
        }

    StopLiveVideo();

    /* Subject responses are stored in the array.  These responses may
       now be written to a file or scored here. */

    /* Free the memory used by the subject clicks */

    FreeMemory(subjectClicks);

    /* reset the current port */

    SetPort(savePort);

    }     /* end CollectData */

/*************************************************************
Function FreeMemory
*************************************************************
1. Purpose: Release memory used to store subject responses.
2. Inputs:  subjectClicks - The array of subject response records.
3. Returned values:
           None.
4. Local variables:
           None.
5. Global variables (and how modified):
           None.
*************************************************************/

static void FreeMemory(subjectClicks)
    SubjectResponse *subjectClicks;
    {

    free(subjectClicks);
    }     /* end FreeMemory */
```

```
/*      Module: ImageDegradation.h

        This module contains the prototypes for functions available in
        the ImageDegradation.c library.

        Prototypes are listed in lexicographical order.

*/


#include "SystemDefines.h"


/* Degrade an image by reducing sampling-grid size */
void BlockyImage(Pixel *sourceImage[IMAGE_HEIGHT], Pixel
*destImage[IMAGE_HEIGHT],
        int sampleSize);
```

```
/*    Module: ImageDegradation.c

      This module contains the routines that will allow an image to be
      degraded.

      Functions are listed in lexicographical order.

*/

#include "MacintoshHeaders.h"
#include "SystemDefines.h"
#include "ImageDegradation.h"

/*****************************************************************************
*
BlockyImage
-----------------------------------------------------------------------------
1. Purpose: Take an IMAGE_HEIGHT x IMAGE_WIDTH image and change its sample
            grid size according to the value specified in sampleSize.  The
            sample size of the image is reduced by averaging blocks of pixels
            under the sample grid.  For example, with a 512 x 512 image and a
            sampleSize value of 32, every 16 x 16 block of pixels are added
            together and the average value is placed into the 16 x 16
            corresponding block in the destination image.  The resulting
            destination image is still 512 x 512, but with only 32 x 32 large
            pixels.
2. Preconditions:
            IMAGE_HEIGHT = IMAGE_WIDTH = 2^n for n = {8, 9, 10}
            sampleSize = 2^n, where 4 <= sampleSize <= IMAGE_HEIGHT
3. Inputs:  sourceImage - The intensity values of the source image.
            sampleSize - The number of pixels the degraded image is to have.
4. Returned values:
            destImage - The intensity values of the resultant image.
5. Local variables:
            blockWidthHeight - The height and width of the large pixel block.
            pixelsPerBlock - Computed once for averaging.
            rowCounter, colCounter - Counter variables.
            subRowCounter, subColCounter - Counter variables.
            totalPixelValue - Sum of the pixels in the large pixel block.
            averagePixelValue - Average of the pixels in the large pixel
            block.
6. Global variables (and how modified):
            None.
*****************************************************************************/

void BlockyImage(sourceImage, destImage, sampleSize)
      Pixel *sourceImage[IMAGE_HEIGHT];
      Pixel *destImage[IMAGE_HEIGHT];
      int sampleSize;
      {
      int blockWidthHeight;
      int pixelsPerBlock;
      register int rowCounter, colCounter;
      register int subRowCounter, subColCounter;
      unsigned int totalPixelValue;
      unsigned int averagePixelValue;

      blockWidthHeight = IMAGE_HEIGHT / sampleSize;
      pixelsPerBlock = blockWidthHeight * blockWidthHeight;
```

```
for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter +=
      blockWidthHeight)
      for (colCounter = 0; colCounter < IMAGE_WIDTH; colCounter +=
            blockWidthHeight)
            {
            totalPixelValue = 0;

for (subRowCounter = rowCounter; subRowCounter < rowCounter +
            blockWidthHeight; subRowCounter++)
            for (subColCounter = colCounter; subColCounter < colCounter +
                  blockWidthHeight; subColCounter++)
                  totalPixelValue +=
                        sourceImage[subRowCounter][subColCounter];

            averagePixelValue = totalPixelValue / pixelsPerBlock;

            for (subRowCounter = rowCounter; subRowCounter < rowCounter +
            blockWidthHeight; subRowCounter++)
            for (subColCounter = colCounter; subColCounter < colCounter +
                  blockWidthHeight; subColCounter++)
                  destImage[subRowCounter][subColCounter] =
                        averagePixelValue;

            }

      }     /* end BlockyImage */
```

```
/*    Module: MacintoshHeaders.h

      This module contains all the #includes for the most commonly used
      declarations during Macintosh programming.  For speed during
      compilation, this header file should be precompiled.

*/


#define _MC68881_       /* enabled for math coprocessor use with the
                           Macintosh II family */

#include    <Quickdraw.h>
#include    <MenuMgr.h>
#include    <FontMgr.h>
#include    <WindowMgr.h>
#include    <TextEdit.h>
#include    <EventMgr.h>
#include    <DeskMgr.h>
#include    <ControlMgr.h>
#include    <ResourceMgr.h>
#include    <DialogMgr.h>
#include    <PackageMgr.h>
#include    <OSUtil.h>
#include    <Color.h>
#include    <ColorToolbox.h>
#include    <ToolboxUtil.h>
#include    <StdFilePkg.h>
#include    <FileMgr.h>
#include    <SlotMgr.h>
#include    <VRetraceMgr.h>
#include    <PrintMgr.h>
#include    "proto.h"          /* include all library function prototypes */
#include    <unix.h>
#include    <stdio.h>
#include    <ctype.h>
#include    <strings.h>
#include    <math.h>
#include    <pascal.h>
#include    <storage.h>
```

```
/*     Module: MiscUtilities.h

       This module contains the prototypes for functions available in
       the MiscUtilities.c library.

       Prototypes are listed in lexicographical order.

*/


#include "SystemDefines.h"


/* Digitize the image on video in port #1 on the NuVision system */
void GrabAnImage(void);

/* Transfer data from the NuVision to the Macintosh system. */
void ReadImageIntensityData(Pixel *theImage[IMAGE_HEIGHT]);

/* Start digitizing the video coming from port #1 and pass it through
   the Pixel Processor */
void StartLiveDisplay(void);

/* End live digitization */
void StopLiveDisplay(void);

/* Output NuVision video in RS170 format */
void SwitchToRS170(void);

/* Continually poll accumulator until a 'white' frame is reached */
void WaitForWhiteFrame(void);

/* Transfer data from the Macintosh into the NuVision system */
void WriteImageIntensityData(Pixel *theImage[IMAGE_HEIGHT]);
```

```
/*    Module: MiscUtilities.c

      This module contains miscellaneous interface and control functions for
      the Macintosh and NuVision systems.

      Functions are listed in lexicographical order.

*/


#include "MacintoshHeaders.h"
#include "SystemDefines.h"
#include "MiscUtilities.h"
#include "nhErrors.h"
#include "Actions.h"
#include "nhBits.h"
#include "Memory.h"
#include "nhGeneral.h"
#include "Main.h"
#include "List.h"
#include "Digitize.h"
#include "nhMEM.h"
#include "nlPixelRows.h"


/* external variables */

extern ImageList outputImages;      /* Images selected as output images */
extern ImageList inputImages;       /* Images selected as input images */


/*************************************************************
Function GrabAnImage
*************************************************************
1. Purpose: Digitize an image from the video in terminal No. 1 on the NuVision
            system and store it in the input image tile selected by the user.
2. Inputs:  None.
3. Returned values:
            None.
4. Local variables:
            grabImage - The input tile selected by the user to grab into.
            byteSwap - TRUE if swap bytes on input, FALSE otherwise.
            currentBusIn - The current data bus grabImage reads from.
5. Global variables (and how modified):
            None.
*************************************************************/

void GrabAnImage()
      {
      Image *grabImage;
      unsigned short currentBusIn;
      Boolean byteSwap;

      /* take all memories off of OPB and AR buses before proceeding */

      nfTakeMEMOffBus(AR_DATA | AR_DATA, OPERAND_SYNC | RESULT_SYNC, 0, NULL);

      /* set up the sync to 480 lines */
```

```
        nhWriteModReg(nvSYN, 0, 1, SYN_R1_PLL_OSC | SYN_R1_PLL_PROG |
              SYN_R1_ALT_FRAME, -1);

        /* set the digitizer to input from the selected input device and
           output on the A result bus. */

        nhWriteModReg(nvDIG, 0, 1, DIG_R1_IN_MONO1 | DIG_R1_AR_ENA, -1);

        /* select the image to grab into */

        grabImage = &(theImages->image[inputImages.image[0]]);

        /* grab image data from the A result bus */

        nhGetMEMInput(grabImage->unit, &currentBusIn, &byteSwap);
        nhSetMEMInput(grabImage->unit, AR_DATA, byteSwap);

        nfHomeMEM (grabImage->unit, grabImage->horizontal, grabImage->column,
              grabImage->row, 1);

        nhSetMEMGrab(grabImage->unit, GRAB_HIGH, 1);

        nhWaitMEMGrab(grabImage->unit);

        DisplayImage (&inputImages);

        }     /* end GrabAnImage */


/*************************************************************
Function ReadImageIntensityValues
*************************************************************
1. Purpose: Take the image data from the NuVision memory space and store
            it in memory on the Macintosh.  Data is taken from the input tile
            that was selected by the user.
2. Inputs:  None.
3. Returned values:
            theImage - Where to store image intensity data from the NuVision
            system.
4. Local variables:
            rowCounter - Counter variable.
5. Global variables (and how modified):
            None.
*************************************************************/

void ReadImageIntensityValues(theImage)
      Pixel *theImage[IMAGE_HEIGHT];
      {
      register int rowCounter;
      Image whichImage;

      whichImage = theImages->image[inputImages.image[0]];

      for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter++)
            nlReadPixels(whichImage.unit, rowCounter, 0, IMAGE_WIDTH, HiBytes,
                  theImage[rowCounter]);

      }     /* end ReadImageIntensityValues */
```

```
/*************************************************************
Function StartLiveDisplay
*************************************************************
1. Purpose: Display the video connected to the video in port No. 1 on the
            NuVision system.  The video is displayed through the output image
            tile selected by the user.  The video is passed through the pixel
            processor's accumulator so the white frame test may be performed.
2. Preconditions:
            An output video tile is selected.
3. Inputs:  None.
4. Returned values:
            None.
5. Local variables:
            grabImage - The output image tile selected by the user to grab
            into byteSwap - TRUE if swap bytes on input, FALSE otherwise.
            currentBusIn - The current data bus grabImage reads from.
6. Global variables (and how modified):
            None.
*************************************************************/

void StartLiveDisplay()
      {
      Image *grabImage;
      unsigned short currentBusIn;
      Boolean byteSwap;

      /* take all memories off of OPB and AR buses before proceeding */

      nfTakeMEMOffBus(OPB_DATA | AR_DATA | OPA_DATA, OPERAND_SYNC |
      RESULT_SYNC, 0, NULL);

      /* set up the sync to 480 lines */

      nhWriteModReg(nvSYN, 0, 1, SYN_R1_PLL_OSC | SYN_R1_PLL_PROG |
      SYN_R1_ALT_FRAME, -1);

      /* set the digitizer to input from video in port #1 and output
         on the A operand bus. */

      nhWriteModReg(nvDIG, 0, 1, DIG_R1_IN_MONO1 | DIG_R1_OPA_ENA, -1);
      nhWriteModReg(nvIPP, 0, 3, IPP_R3_OUTPUT_ENA, -1);

      /* OR the output video so it passes through unaltered */

      nhWriteModReg(nvIPP, 0, 7, IPP_R7_OR, -1);
      nhWriteModReg(nvIPP, 0, 11, IPP_R11_OR_1 | IPP_R11_OR_2, -1);

      grabImage = &(theImages->image[outputImages.image[0]]);

      /* video comes out of the pixel processor on the A result bus */

      nhGetMEMInput(grabImage->unit, &currentBusIn, &byteSwap);
      nhSetMEMInput(grabImage->unit, AR_DATA, byteSwap);

      nfHomeMEM(grabImage->unit, grabImage->horizontal, grabImage->column,
            grabImage->row, 1);
```

```
      nhSetMEMGrab(grabImage->unit, GRAB_HIGH, 0);

      DisplayLive(&outputImages);

      }           /* end StartLiveDisplay */


/*************************************************************
Function StopLiveDisplay
*************************************************************
1. Purpose: End live video display.  The frame grab in process is completed
            and the result stored in the output video tile selected by the
            user.
2. Inputs:  None.
3. Returned values:
            None.
4. Local variables:  grabImage - The output image tile selected by the user to
            grab into.
5. Global variables (and how modified):
            None.
*************************************************************/

void StopLiveDisplay()
      {
      Image *grabImage;

      grabImage = &(theImages->image[outputImages.image[0]]);

      nhSetMEMGrab(grabImage->unit, GRAB_HIGH,1);

      nhWaitMEMGrab(grabImage->unit);

      } /* end StopLiveDisplay */


/*************************************************************
Function SwitchToRS170
*************************************************************
1. Purpose: Switch the output video signal to RS170 format.  Note: The
            hardware jumper change from RGB to RS170 format described in the
            NuVision must be performed prior to calling this function.
2. Inputs:  None.
3. Returned values:
            None.
4. Local variables:
            None.
5. Global variables (and how modified):
            None.
*************************************************************/

void SwitchToRS170()
      {

      nhWriteModReg(nvDIS, 0, 2, DIS_R2_RGB_DIS | DIS_R2_OLAY_ENA ,
            DIS_R2_RGB_DIS | DIS_R2_OLAY_ENA);

      }     /* end SwitchToRS170 */
```

```
/*************************************************************
Function WaitForWhiteFrame
*************************************************************
1. Purpose: Continually poll all incoming video looking for a shift from a
            series of black (gray scale = 0) frames to a white (gray scale =
            255) frame.  The accumulator reads 16 bits per pixel instead of 8,
            therefore all 8 bit gray scale intensities are left shifted 8
            places prior to summation.  When the accumulated value of all
            pixel intensities reaches 8.589 x 10^9, the function will
            complete.  This value corresponds to a 512 x 512 image with all
            pixels set to 128 intensity.  Note: This function will run to
            infinity if a white frame is not encountered.
2. Preconditions:
            This function is hard-coded for a 512x512 image only.
3. Inputs:  None.
4. Returned values:
            None.
5. Local variables:
            done - TRUE when the white frame has been encountered, FALSE
            otherwise.
            registerContents - Contents of register 15 on the pixel processor.
            Holds the most significant bits of the accumulation.
6. Global variables (and how modified):
            None.
*************************************************************/

void WaitForWhiteFrame()
      {
      unsigned short registerContents;
      Boolean done = FALSE;

      WatchCursor();

      /* Keep checking accumulator for a value greater than or equal to 8.589
      x 10^9.  This is our 'white frame' indicator.  The value corresponds to
      a 512 x 512 image with all pixels set at 128 intensity. */

      while (!done)
            {
            nhWriteModReg(nvIPP, 0, 1, IPP_R1_NEXT_INTRVL | IPP_R1_GO |
                  IPP_R1_ACCUM_ENA, IPP_R1_NEXT_INTRVL | IPP_R1_GO |
                  IPP_R1_ACCUM_ENA);
            nhReadModReg(nvIPP, 0, 15, &registerContents);
            if (registerContents >= 0x0002)
                  done = TRUE;
            }

      InitCursor();
      SysBeep(1);

      } /* end WaitForWhiteFrame */
```

```
/************************************************************
Function WriteImageIntensityValues
************************************************************
1. Purpose: Take the image data from the Macintosh memory space and write it
            into the memory space of the NuVision system.  Data is written to
            the output tile selected by the user.
2. Inputs:  theImage - The image intensity data to write to the NuVision
            system.
3. Returned values:
            None.
4. Local variables:
            rowCounter - Counter variable.
            whichImage - Which image tile on the NuVision system in which
                  to write the image data.
5. Global variables (and how modified):
            None.
************************************************************/

void WriteImageIntensityValues(theImage)
      Pixel *theImage[IMAGE_HEIGHT];
      {
      register int rowCounter;
      Image whichImage;

      whichImage = theImages->image[outputImages.image[0]];

      for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter++)
            nlWritePixels(whichImage.unit, rowCounter, 0, IMAGE_WIDTH,
            HiBytes, theImage[rowCounter]);


      }     /* end WriteImageIntensityValues */
```

```
/*    Module: SystemDefines.h

      This module contains the definitions for the screen size and the
      data type for a pixel.

*/


#define NULL            0L          /* NULL is a 0 pointer */

#define IMAGE_HEIGHT    512         /* number of pixels per column */
#define IMAGE_WIDTH     512         /* number of pixels per row */


typedef enum { false, true, FALSE = 0, TRUE } Boolean;

typedef unsigned char Pixel;        /* pixels are 8 bit gray scale */
```

```
/*      Module: VCRControl.h

        This module contains the prototypes for functions available in the
        VCRControl.c library.  Also, the assignment of VCRs to serial ports is
        made here.  Currently, the playing VCR must be connected to the printer
        serial port and the recording VCR must be connected to the modem serial
        port.

        Prototypes are listed in lexicographical order.

*/


#include    <SerialDvr.h>

#define     PLAY_VCR    sPortB      /* The playing VCR needs to be connected
to the
                                       printer port. */
#define     RECORD_VCR  sPortA      /* The recording VCR needs to be connected
to the
                                       modem port. */


/* Advance the specified VCR by one frame */
void AdvanceVCR(SPortSel whichVCR);

/* Close all serial ports connected to the VCRs */
void CloseVCRs(void);

/* Open all serial ports connected to the VCRs
void InitializeVCRs(void);
```

```
/*    Module: VCRControl.c

      This module contains routines that control the VCRs connected to the
      Macintosh serial ports.

      Functions are listed in lexicographical order.

*/


/* VCR/serial port defines */

#define MODEM_PORT_ID          -7    /* reference number for modem port */
#define PRINTER_PORT_ID        -9    /* reference number for printer port */
#define DTR_ON                 17    /* DTR pin activate signal */
#define DTR_OFF                18    /* DTR pin deactivate signal */


#include "MacintoshHeaders.h"
#include <SerialDvr.h>
#include "VCRControl.h"



/************************************************************
Function AdvanceVCR
************************************************************
1. Purpose: Advance the specified VCR by one frame.  The DTR signal is toggled
            between on and off.  This action directs the VCR control circuit
            to short the 1 shot record terminal on the VCR causing it to
            advance 1 frame.
2. Inputs:  whichVCR - Which VCR to advance, the playing VCR (PLAY_VCR) or the
            recording VCR (RECORD_VCR).
3. Returned values:
            None.
4. Local variables:
            None.
5. Global variables (and how modified):
            None.
************************************************************/
void AdvanceVCR(whichVCR)
      SPortSel whichVCR;
      {
      switch(whichVCR)
            {
            case PLAY_VCR:
                  Control(MODEM_PORT_ID, DTR_ON, NULL);
                  Control(MODEM_PORT_ID, DTR_OFF, NULL);
                  break;
            case RECORD_VCR:
                  Control(PRINTER_PORT_ID, DTR_ON, NULL);
                  Control(PRINTER_PORT_ID, DTR_OFF, NULL);
                  break;
            default:
                  break;
            }

      }     /* end AdvanceVCR */
```

```
/************************************************************
Function CloseVCRs
************************************************************
1. Purpose: Close the modem and printer serial ports connected to the
            VCR control circuit.
2. Inputs:  None.
3. Returned values:
            None.
4. Local variables:
            None.
5. Global variables (and how modified):
            None.
************************************************************/
void CloseVCRs()
      {

      RAMSDClose(PLAY_VCR);
      RAMSDClose(RECORD_VCR);

      }     /* end CloseVCRs */


/************************************************************
Function InitializeVCRs
************************************************************
1. Purpose: Open the modem and printer serial ports connected to the
            VCR control circuit.
2. Inputs:  None.
3. Returned values:
            None.
4. Local variables:
            None.
5. Global variables (and how modified):
            None.
************************************************************/
void InitializeVCRs()
      {

      RAMSDOpen(PLAY_VCR);
      RAMSDOpen(RECORD_VCR);

      }     /* end InitializeVCRs */
```

```
/*     Module: VideoGeneration.h

       This module contains the prototypes for functions available in
       the VideoGeneration.c library.

       Prototypes are listed in lexicographical order.

*/


/* Advance the playing VCR frame by frame, process each frame, and
   record the result frames on another VCR. */
void GenerateVideo(int numberOfFrames, int sampleSize);
```

```
/*    Module: VideoGeneration.c

      This module contains the routines that control the production of
      simulated real time video.

      Functions are listed in lexicographical order.

*/


#include "MacintoshHeaders.h"
#include "SystemDefines.h"
#include "VideoGeneration.h"
#include "MiscUtilities.h"
#include "ImageDegradation.h"


/* Local function prototypes (in lexicographical order): */

static void AllocateMemory(Pixel *inputImage[IMAGE_HEIGHT],
      Pixel *outputImage[IMAGE_HEIGHT]);

static void FreeMemory(Pixel *inputImage[IMAGE_HEIGHT],
      Pixel *outputImage[IMAGE_HEIGHT]);


/************************************************************
Function AllocateMemory
************************************************************
1. Purpose: To allocate memory in the Macintosh to store 2 images, one for
            input the other for output, of size IMAGE_HEIGHT x IMAGE_WIDTH.
            Each array element stores one Pixel.
2. Inputs:  None.
3. Returned values:
            inputImage - Memory space for storing the input image.
            outputImage - Memory space for storing the result image.
4. Local variables:
            rowCounter - Counter variable.
5. Global variables (and how modified):
            None.
************************************************************/

static void AllocateMemory(inputImage, outputImage)
      Pixel *inputImage[IMAGE_HEIGHT], *outputImage[IMAGE_HEIGHT];
      {
      register int rowCounter;

      for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter++)
            inputImage[rowCounter] = (Pixel *)malloc(sizeof(Pixel) *
            IMAGE_WIDTH);

      for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter++)
            outputImage[rowCounter] = (Pixel *)malloc(sizeof(Pixel) *
            IMAGE_WIDTH);

      }     /* end AllocateMemory */
```

```
/*************************************************************
Function FreeMemory
*************************************************************
1. Purpose: To release all memory used for image storage.
2. Inputs:  inputImage - Memory for the original image storage.
            outputImage - Memory for the result image storage.
3. Returned values:
            None.
4. Local variables:
            rowCounter - Counter variable.
5. Global variables (and how modified):
            None.
*************************************************************/

static void FreeMemory(inputImage, outputImage)
      Pixel *inputImage[IMAGE_HEIGHT], *outputImage[IMAGE_HEIGHT];
      {
      register int rowCounter;

      for (rowCounter = 0; rowCounter < IMAGE_HEIGHT; rowCounter++)
            {
            free(inputImage[rowCounter]);
            free(outputImage[rowCounter]);
            }

      }     /* end FreeMemory */


/*************************************************************
Function GenerateVideo
*************************************************************
1. Purpose: Allocate memory in the Macintosh to store 2 images, one for input
            the other for output, of size IMAGE_HEIGHT x IMAGE_WIDTH.
2. Preconditions:
            An output and input image tile must be selected.
3. Inputs:  numberOfFrames - Number of frames to process.
            sampleSize - Degrade the image by reducing its sampling grid size
            to this amount.
4. Returned values:
            None.
5. Local variables:
            frameCounter - Counter variable.
            inputImage - The original, unprocessed image.
            outputImage - The processed image.
6. Global variables (and how modified):
            None.
*************************************************************/

void GenerateVideo(numberOfFrames, sampleSize)
      int numberOfFrames;
      int sampleSize;
      {
      int frameCounter = 1;
      Pixel *inputImage[IMAGE_HEIGHT], *outputImage[IMAGE_HEIGHT];

      SwitchToRS170();

      InitializeVCRs();
```

```
AllocateMemory(inputImage, outputImage);

while (frameCounter <= numberOfFrames)
      {
      /* digitize an image into the NuVision system */

      GrabAnImage();

      /* read image into Macintosh memory from the NuVision system */

      ReadImageIntensityValues(inputImage);

      /* advance playing VCR to next frame */

      AdvanceVCR(PLAY_VCR);

      /* Perform operations on input image */

      BlockyImage(inputImage, outputImage, sampleSize);

      /* display the resultant image by writing it into the NuVision
      memory */

      WriteImageIntensityValues(outputImage);

      /* advance recording VCR to record result image */

      AdvanceVCR(RECORD_VCR);

      ++frameCounter;

      }     /* end while */

FreeMemory(inputImage, outputImage);

CloseVCRs();

                        }     /* end GenerateVideo */
```